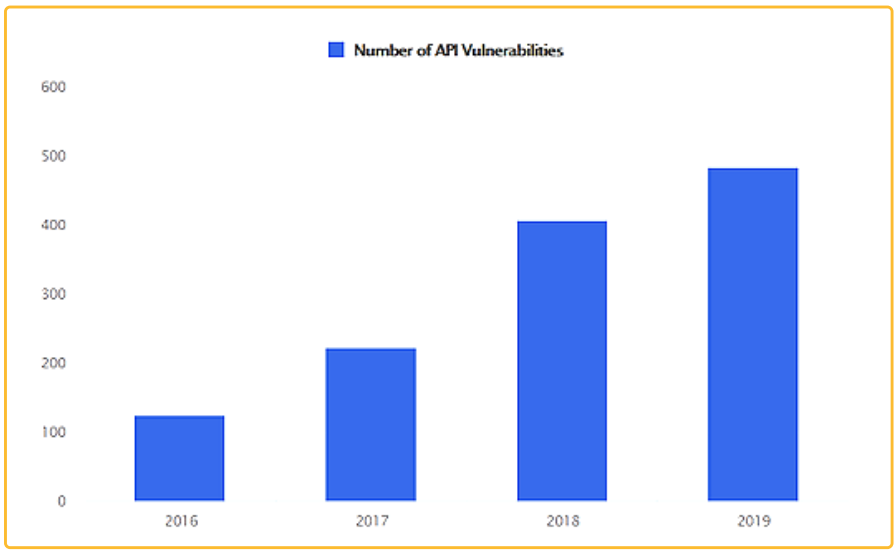
astra

The Ultimate API Security Audit & Penetration Testing Checklist

API (Application Programming Interface) has been around for a very long time. Attributing to its wide usage, it became an easy vector for hackers. The vulnerabilities of API can lead to security failure, data breach, unauthenticated access, and so on. Further, a vulnerable API can cost a company millions of dollars if it goes unchecked.

As you can see in the following bar graph, the number of vulnerabilities has constantly been increasing over the years - a growth of 18.9% in the number of API vulnerabilities since 2018.

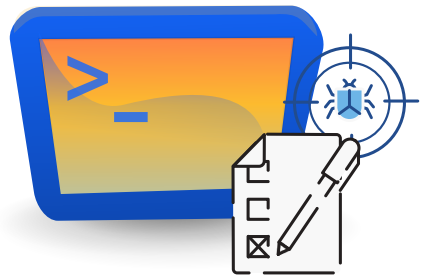


Besides the rising API vulnerabilities, the security of sensitive information handled by API is another big reason to get its security tested.

For a business, the need for an API Security Testing is getting direr with every passing day. This checklist document guides API developers as well as internal security teams on how to attain the maximum level of protection for their API and the sensitive data stored/processed within, by conducting an effective security audit.

A vulnerability assessment & penetration testing checklist for API security will ensure that you don't miss any crucial area of your API services and ensure they are configured correctly with the highest level of security.

API Security Audit Checklist



Input validation

1. Do not trust input parameters/objects.
2. Validate input: length / range / format and type.
3. Achieve an implicit input validation by using strong types like numbers, booleans, dates, times or fixed data ranges in API parameters.
4. Constrain string inputs with regexps.
5. Reject unexpected/illegal content.
6. Make use of validation/sanitation libraries or frameworks
7. Define an appropriate request size limit and reject requests exceeding the limit with HTTP response status 413 Request Entity Too Large.
8. Use a secure parser for parsing the incoming messages. If you are using XML, make sure to use a parser that is not vulnerable to **XXE** and similar attacks.
9. Use the proper HTTP method according to the operation: GET (read), POST (create), PUT/PATCH (replace/update), and DELETE (to delete a record), and respond with *405 Method Not Allowed* if the requested method isn't appropriate for the requested resource.
10. Validate user input to avoid common vulnerabilities (e.g. XSS, SQL-Injection, Remote Code Execution, etc.).
11. Don't use any sensitive data (credentials, Passwords, security tokens, or API keys) in the URL, but use standard Authorization header.
12. Use an API Gateway service to enable caching, Rate Limit policies (e.g. Quota, Spike Arrest, or Concurrent Rate Limit) and deploy APIs resources dynamically.

Content type validation

1. Reject requests containing unexpected or missing content type headers with HTTP response status 406 Unacceptable or 415 Unsupported Media Type.
2. For XML content types ensure appropriate XML parser hardening, see the **XXE cheat sheet**.
3. Avoid accidentally exposing unintended content types by explicitly defining content types e.g. ***Jersey*** *(Java) @consumes("application/json"); @produces("application/json")*. This avoids XXE-attack vectors for example.
4. Do NOT simply copy the Accept header to the Content-type header of the response.
5. Reject the request (ideally with a 406 Not Acceptable response) if the Accept header does not specifically contain one of the allowable types.
6. Ensure sending intended content type headers in your response matching your body content e.g. application/json and not application/javascript.

Authentication

1. Don't use Basic Auth. Use standard authentication instead (e.g. JWT, OAuth).
2. Don't reinvent the wheel in Authentication, token generation, password storage.
3. Use Max Retry and jail features in Login.
4. Use encryption on all sensitive data.

JWT (JSON Web Token)

1. Use a random complicated key (JWT Secret) to make brute forcing the token very hard.
2. Don't extract the algorithm from the header. Force the algorithm in the backend (HS256 or RS256).
3. Make token expiration (TTL, RTTL) as short as possible.
4. Don't store sensitive data in the JWT payload, it can be decoded easily.
5. Ensure JWTs are integrity protected by either a signature or a MAC.
6. Do not allow the unsecured JWTs: {"alg":"none"}.
7. In general, signatures should be preferred over MACs for integrity protection of JWTs.

Management Endpoints

1. Avoid exposing management endpoints via Internet.
2. If management endpoints must be accessible via the Internet, make sure that users must use a strong authentication mechanism, e.g. multi-factor.
3. Expose management endpoints via different HTTP ports or hosts preferably on a different NIC and restricted subnet.
4. Restrict access to these endpoints by firewall rules or use of access control lists.

API Keys

1. Enforce API keys for every request to the protected endpoint.
2. Return 429 Too Many Requests HTTP response code if requests are coming in too quickly.
3. Revoke the API key if the client violates the usage agreement.
4. Do not rely exclusively on API keys to protect sensitive, critical or high-value resources.

Error Handling

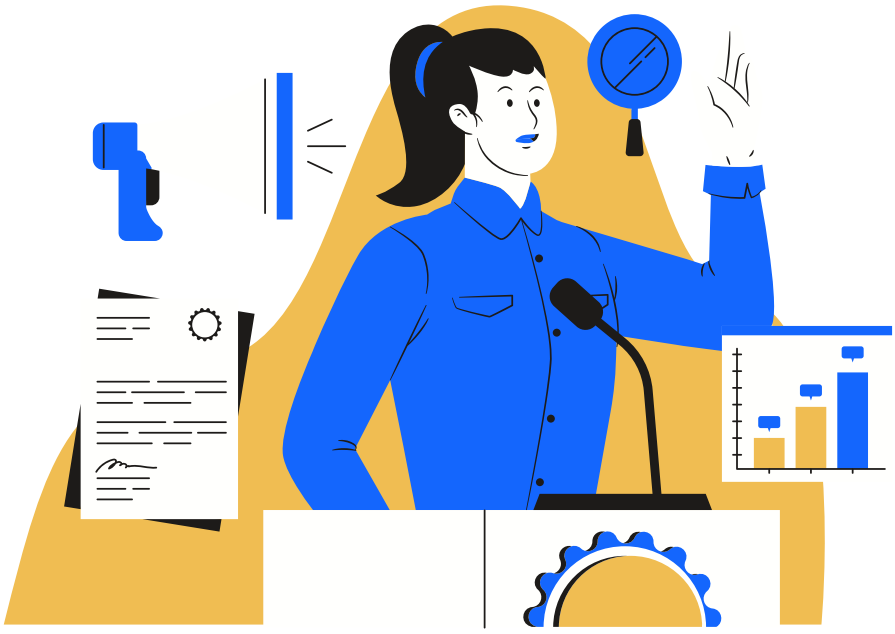
1. Respond with generic error messages - avoid revealing details of the failure unnecessarily.
2. Do not pass technical details (e.g. call stacks or other internal hints) to the client.

Audit Logs

1. Write audit logs before and after security related events.
2. Consider logging token validation errors in order to detect attacks.
3. Take care of log injection attacks by sanitising log data beforehand.

API Output Checks

1. Send X-Content-Type-Options: nosniff header.
2. Send X-Frame-Options: deny header.
3. Send Content-Security-Policy: default-src 'none' header.
4. Remove fingerprinting headers - X-Powered-By, Server, X-AspNet-Version, etc.
5. Force content-type for your response. If you return application/json, then your content-type response is application/json.
6. Don't return sensitive data like credentials, Passwords, or security tokens.
7. Return the proper status code according to the operation completed. (e.g. 200 OK, 400 Bad Request, 401 Unauthorized, 405 Method Not Allowed, etc.).



API Penetration Testing Checklist

- [x] Observe each parameter in every module of API, understand how the data is transferred from source to destination. Try to play with the parameter by tampering them.

- [x] Identify if the API has any authorization token if it is having then remove that authorization token and see application response. In some cases, if authorization is not implemented correctly then API might give you access to forbidden assets of application.

- [x] Analyze and check each module with a different access level of user ex: admin, moderator, normal user.

- [x] Check whether admin modules can be accessed via the restricted user.

- [x] Identify the parameters which may vulnerable to IDOR type vulnerabilities such as id=1234 and also look at the cookies for manipulating the Ids.

- [x] Check injection vulnerabilities by inserting special characters in all parameters in a request and check the response from the server. If you find any stack traces then use the information for further exploitation.

- [x] Insert greater than, less than (<,>) characters in all parameters and see response whether the application encoding them as > and <. If an application doesn't escape any special characters then the application may be vulnerable to client-side attacks such as XSS (cross-site scripting).

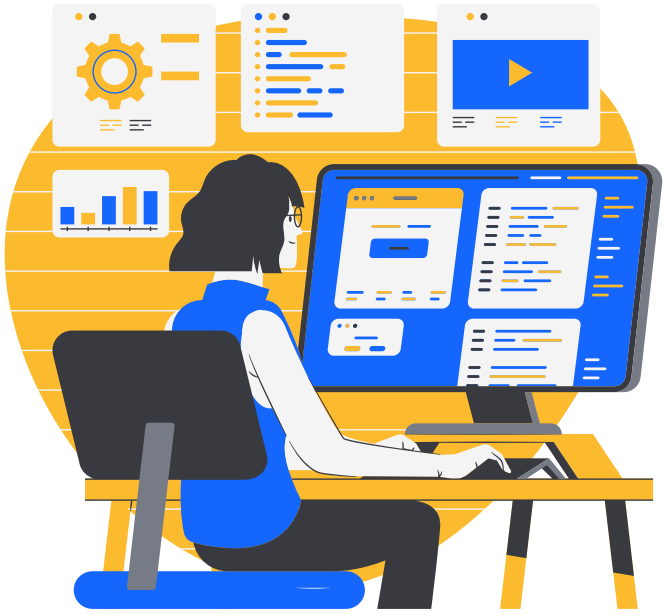
- [x] Modify the content-type server header for understanding the XML entity injection attack. Example: change content Application/JSON to application/XML and insert the XML entity payload to find the XML entity injection.

- [x] Test for API Input Fuzzing (for this you can use an open-source fuzzing tool called **Fuzzapi**)

- [x] Test for API Injection Attacks:
  - Test for SQL Injection (using a tool called **SQLmap**)
  - Test for Command Injection (using a tool called **Commix**)
  - Test for Parameter Tampering
  - Test for Unhandled HTTP Methods

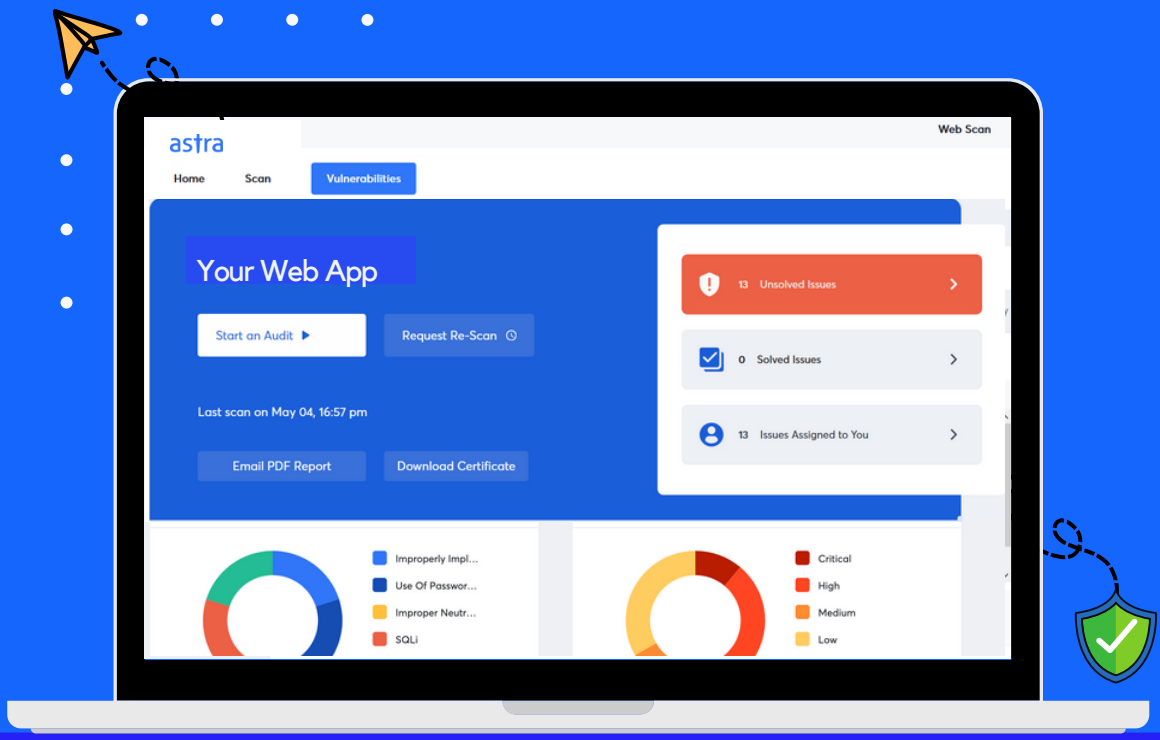
- [x] Test for File Upload vulnerability
- [x] Test for XML Bomb (DoS)
- [x] Test for Session fixation
- [x] Test for XML Signature wrapping
- [x] Test for Host Cipher Support/ Valid Certificate/ Protocol Support
- [x] Test for Authentication based attacks
- [x] Test for Replay attacks

Tools: **Fiddler, Burp Suite, Acunetix/IBM Security, AppScan, ZAP Proxy, Curl, SOAP UI, etc.**



References: OWASP
https://github.com/shieldfy/API-Security-Checklist

Looking for a professional Security Audit & VAPT for your APIs? Astra Security can help.



Astra Security's vulnerability management dashboard comes with a birds eye view for management - keeping you always on the top of security assessment status and help you achieve regulatory compliance such as SOC2, PCI-DSS, ISO, & more. Plus, turn your SecOps into DevSecOps with CI/CD integrations.

Video PoCs, selenium scripts & collaboration with security team enables your developers to fix the vulnerabilities in record time.

With Astra Security, Pentest takes 40% less time than other solutions.

Pentesting based on industry leading practices such as OWASP, OSSTMM, WASC, CREST, NIST etc.

astra

Contact us to get a free demo

www.getastra.com | Schedule a Call | hello@getastra.com

fb.com/getAstra | @getastra | linkedin.com/company/getastra